

# Design Strategy

Rebecca Wirfs-Brock

March 4, 2008

“The essence of strategy is that you must set limits on what you're trying to accomplish.”—Michael Porter

Software designers and managers can find it challenging to agree upon the “sweet spots” of their system that warrant their best design efforts. Most projects are short on time, budget, and resources. How can you stay ahead of the design curve and where should you focus your design energies to gain the most leverage?

## Start Sharing Assumptions

Throughout my engineering career I've used a variety of techniques to organize my thoughts—from private journaling to list making to sketching out challenges ahead. Experienced designers are expected to devise good solutions despite uncertainty, conflicting priorities, and limited time. Early in most design efforts we typically don't know enough to make informed decisions or predict accurately what the hard parts are going to be.

Over time I've improved my ability to anticipate problems. But I still find it useful to set the stage for design work by writing about my hopes, wishes, fears and convictions in a couple of short paragraphs that state how I see the situation. In this short designer's story I identify things I know with certainty, things that need to be explored, or any nagging concerns I might have. People may have been talking about what's important, but in my story I get to put my own spin on things. I can make bold statements about what's notable, what I think we should focus on, and what I suspect will be easy or difficult.

I used to write these stories and tuck them away. While writing them helped me collect my thoughts, no one else knew what I was thinking! And I didn't get a whiff of others' concerns or confident assertions until much later, either (sometimes too late).

Collectively as a team we had to stumble upon each others' hidden assumptions and aspirations. Today I find that collaborative design efforts get off on a better footing if teammates air their thoughts about their design future in a non confrontational setting. Whenever I have the chance I encourage teams to kick off new design efforts or major iterations by writing and sharing their initial design thoughts.

Even the most reluctant developer who only wants to hack code can bang out a designer's story if it is short, sweet, to the point, and takes only 15 minutes. They usually go along if I suggest it'll be good for the team to air their unedited initial thoughts on the design.

After writing stories, we read them out loud. It's remarkable how this simple activity can help a team gel and gain a collective perspective on the nature of the work that lies ahead. And it also provides an opportunity for those perennially quiet voices to be heard.

Although we can stop after merely writing and sharing stories, I sometimes find it useful to use them as “seed corn” to start our design planning. Perhaps we might make a candidate list of design hotspots where we think we’ll need to focus or even an “is it really as easy as that?” list to verify with project stakeholders.

### **Balance Design Efforts**

Being armed with a toolkit of design techniques and practices doesn’t preclude unpredictable challenges or unexpected twists. To keep focused I find it useful to fit day-to-day design and development work into these categories:

- *Core* design work. The core is the core because without it there is no reason to build the rest.
- *Revealing* design problems. Revealing problems, when pursued, lead to a fundamental, deep understanding about the nature of your software. Just because some part is difficult or tricky, however, isn’t enough to make it revealing.
- The *rest*. Although not trivial, the rest requires hard work and attention to detail but far less creativity or inspiration.

Each problem type warrants a different kind of energy and has a different work rhythm. Core problems must be solved. Your software won’t meet users’ needs or stand up to the rigors of use without a well-designed core. This is engineering at its best. Designing the core requires energy and focused attention. It requires steady, consistent consideration.

Revealing problems can be squishy and hard to characterize or even know when they are completely solved. Each time you dig deeper into a revealing problem you learn something new. They can’t always be resolved in tidy ways. They deserve special attention and can derail the best laid plans and intentions.

Working on revealing problems often involves periods of intense concentration, design, reflection, testing, and implementation, interspersed with open, honest communication about progress. Sometimes revealing problems can cause you to completely shift your view and discard what you had assumed to be fundamental truths about your design. Management cringes when you expose a revealing problem because the path to their solution is always unpredictable.

The rest includes mundane, tedious, or mildly interesting design work. An overabundant supply is always present, pressing, and willing to soak up every spare cycle on most projects. The challenge is to avoid getting bogged down in the rest—or slighting it as mere “grunt work”. If the bulk of your code base isn’t core you still don’t want it to drag your project down. Tactically there may not be very many strategic design decisions in the rest that require heavy mental lifting, but it needs attention. Developing and promoting “standard ways” of doing things—recipes or patterns of practice, if you will—can take the guess work out of design of the rest.

### **Tackle Core Design Problems**

The key to balancing core design work with other design activities is to ensure that the core is well known, solid, understood, and nurtured. Core design work requires constant attention and best efforts. But what exactly is core? Core problems include those fundamental aspects of your design (no, not every part can be fundamental) that are essential to your software's success. It might be the design of error handling and recovery, key domain objects, critical algorithms, performance optimization mechanisms, security, optimized data retrieval, or any number of other aspects of your system.

Eric Evans (*Domain Driven Design*, Addison-Wesley, 2003) claims that most teams who develop complex products or enterprise-wide application give short shrift to domain models. He's found that developers often discount domain models and business logic, instead preferring to prove their design mettle by working through the tough technical infrastructure bits. While getting infrastructure in place is important, Eric argues, it isn't what is going to distinguish your software over the long run. Eric suggests we remedy this by, "Find[ing] the core domain and provide a means of easily distinguishing it from the mass of supporting model and code." He suggests making the domain core small and bringing top design talent to bear on designing and building it. To do so effectively, developers, need to interact and develop deep knowledge by working directly with domain experts—something that is often out of their comfort zone. Although he makes no claims that finding and agreeing on the core domain is easy, once consensus is reached, he suggests it will make most subsequent design decisions easier. Whenever a choice needs to be made between alternatives, choose the one that has the least negative impact or most positively affects the core.

My experience developing complex systems has varied from Eric's. Sometimes I find domain-specific behaviors to be at the heart of core design problems, sometimes there isn't much domain-specific behavior. So there isn't much reason to develop a rich domain model (in which case I usually say so and suggest we move on to more pressing design concerns). With most complex systems, however, it isn't easy to isolate a single area for concentrated design focus. There may be several "core" problems and a slew of important design constraints that need to be considered.

Nick Rozanski and Eoin Woods (*Software Systems Architecture: Working With Stakeholders Using Viewpoints and Perspectives*, Addison-Wesley 2005) suggest a relatively straightforward process for focusing on architecturally significant design aspects. They advise designers and architects to engage in conversations with system stakeholders to develop consensus on the relative importance of desired system functions and qualities such as reliability, scalability or usability. This prioritized set of concerns can then drive appropriate strategic design, modeling and prototyping activities.

It's hard to optimize the design effort for complex, multi-faceted systems. We must balance conflicting priorities and exercise a wide range of design skills, techniques, and tactics. Iterative development enables us to focus our attention on areas that present significant design risk and establish design baselines that can be built upon. But even when applying these techniques we have to sharpen our focus and apply design energy to

where we think it will have the most impact. It's not possible to apply the same level of effort everywhere.